NASA TM-83373

NASA Technical Memorandum 83373

NASA-TM-83373 19830021326

Design of a High-Speed Digital Processing Element For Parallel Simulation

Edward J. Milner and David S. Cwynar
Lewis Research Center
Cleveland, Ohio

June 1983

NASA

DESIGN OF A HIGH-SPEED DIGITAL PROCESSING ELEMENT FOR PARALLEL SIMULATION

Edward J. Milner and David S. Cwynar
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

SUMMARY

A prototype of a custom-designed computer to be used as a processing element in a multi-processor-based jet engine simulator is described in this report. The computer was custom-designed to give it the speed and versatility required to simulate a jet engine in real-time. Real-time simulations are needed for closed-loop testing of digital electronic engine controls. The prototype computer has a microcycle time of 133 nanoseconds. This speed was achieved by: (1) prefetching the next instruction while the current one is executing, (2) transporting data using high-speed data busses, and (3) using state-of-the-art components such as a VLSI multiplier. However, some other features usually found in commercially available computers, but not necessarily required in a simulator, were left out of the custom design to reduce cost and system complexity. These include complex interrupt structures, byte addressability, and a large memory addressing range.

The report discusses processing element requirements, design philosophy, the architecture of the custom-designed processing element, the comprehensive instruction set, the diagnostic support software, and the development status of the custom design. Problem areas encountered in designing the processing element, along with logic circuitry used to eliminate those problems, are pointed out. By doing so, the authors hope that the reader may gain some insight into the kinds of difficulties that might be expected when undertaking the development of a custom-designed computer to meet special application needs.

INTRODUCTION

Even though the rapid growth in microelectronic technology in recent years has made it possible to build compact high-speed digital computers, accurate real-time digital simulations of modern airbreathing engines and their controls are still difficult to achieve. Because of the large number of complex nonlinear calculations required in such a simulation and the sequential nature of digital computers, real-time execution of these calculations requires simplification of the engine model and/or the use of an extremely fast, dedicated mainframe computer (e.g., ref. 1).

The technique of parallel processing seems to show promise for real-time simulation. Several computers concurrently operating on different portions of the simulation can effectively reduce the computation time and may provide real-time response of the simulation (ref. 2). Parallel computation may permit real-time execution of simulations heretofore considered too complex for such execution speeds. The use of low-cost mini or microcomputers can make this

N83-29597 #

approach cost-effective relative to current approaches (i.e., use of mainframe computers).

The design of a real-time digital simulator (RTDS), to be used as a research tool for the development and testing of airbreathing engines and their controls, is being pursued at the NASA Lewis Research Center (LeRC). A conceptual design of a possible Lewis RTDS is presented in reference 3. A block diagram of this design is shown in figure 1.

Basically, the RTDS consists of several processing elements (PE's) synchronized on a high-speed data transfer bus by a transfer controller. All but two of the processing elements can be used to perform concurrent simulation computations. One of the two remaining processing elements is dedicated to input/output functions. The last processing element serves as the real-time-extension (RTX) of the front-end-processor (FEP). It is a special purpose processor linking low-speed operator interaction with the high-speed simulator core. The FEP provides the interface between the operator and simulator that permits the operator to control the simulation execution. The FEP, based on the Motorola MC68000 microprocessor, handles such functions as peripheral communications and mode control. In addition, a host computer interface allows the downlinking and uplinking of data and programs between a host computer and the simulator. At LeRC the host computer is an International Business Machine (IBM) 370/3033.

This report describes the design of a prototype computer meant to be used as a processing element (PE) in a multi-processor-based jet engine simulator. The custom design provides computation speeds and programming flexibility not obtainable using commercially available computers. The report discusses the requirements for a PE to be used in a RTDS, the Lewis design philosophy, the architecture of the custom-designed computer and its support software, and the development status of the custom design.

PROCESSING ELEMENT REQUIREMENTS

The custom-designed processing element (PE) was required to satisfy several basic requirements. The first and foremost requirement was computational speed. The kinds of engine control research for which the simulator would be used require the simulation calculations to be performed in less than 10 milliseconds. This requirement pointed to a maximum microcycle time of 250 nanoseconds. Since the PE was to be used not only as the principal computing element in the simulator but also as a tool in future hardware and software research, every effort was made to minimize the microcycle time.

Another requirement was that the PE be versatile and convenient to use. The PE must be flexible enough to accommodate both hardware and software changes. Because the simulator is intended to be a research tool, the PE must be compatible with different simulator hardware configurations. The PE design must allow changes to its instruction set to evaluate its effects on system operation.

Because the PE is to be used in a parallel processing system, the PE memory access must be sufficiently fast. Otherwise, a disproportionate amount of time will be required to transfer the data among the elements comprising the simulator. Finally, the prototype design must be inexpensive to build.

PROCESSOR OVERVIEW

The custom-designed computer presented in this report satisfies all of these requirements. Speed was acquired by building the PE with state-of-the-art (VLSI multiplier, e.g.) and employing innovative circuitry to reduce the execution time. As will be seen later, the prototype design, with its microcycle time of 133 nanoseconds, is capable of executing a sequence of code two to three times faster than a Digital Equipment Corporation (DEC) PDP 11/70 computer. In addition the custom-designed PE provides the programmer with a very powerful and complete instruction set for simulating jet engine behavior.

Almost 300 different instructions, each with an easy to understand natural mnemonic name, cover the spectrum of arithmetic, logical, and conditional operations. Many operations which would normally require multiple instructions on other computers are available on this custom design as a single command. Selecting the maximum of two values, MAX, for example, is a single instruction on the custom-designed PE.

Instructions are not hardwired in the custom-designed PE, but reside in the PE as microcode. No hardware changes are required to modify the instruction set. In addition, all circuit connections on all circuit boards are wirewrap connections. Thus, soldering is unnecessary when modifying wiring on the computer boards.

The PE design will permit the transfer of data between PE's to be carried out quickly and orderly in the simulator, the process consuming an acceptable portion of the computation update cycle. The hardware meets this high-speed data transfer requirement by using pipelining, 45 nanosecond memory, and Schottky TTL components. Costs were kept to a minimum by building the prototype PE inhouse at LeRC. The hardware for the system cost approximately $11 000. This includes the chassis, circuit boards, components, wire, etc. for the PE; it does not include any support hardware or diagnostic equipment. A photograph of the prototype hardware is included as figure 2.

Salient features of the design include: a 133 nanosecond microcycle time; an advanced microcycle controller to minimize the number of cycles associated with instructions involving conditionals; an enhanced instruction set that permits rapid execution of simulation-related functions (SELECT MAX or SELECT MIN executed in 166 nsec, for example), and a very-large-scale integrated (VLSI) 16-bit multiplier that permits three different types of multiplication to be executed in 400 nanoseconds each.

DESIGN PHILOSOPHY

The emergence of moderately priced, very high speed metal-oxide-semiconductor (MOS) and transistor-transistor-logic (TTL) memories has made possible the design of moderately-priced memory modules with access times of 80 to 100 nanoseconds. As a result, traditional minicomputer design guidelines, which assume that the arithmetic logic unit (ALU) is at least twice as fast as the memory, have become obsolete. Therefore, to achieve ALU cycle times that are compatible with the speeds of these advanced memories, one could use the latest emitter-coupled-logic (ECL) bit-slice devices. However, these devices are very expensive, consume a considerable amount of power, and require special circuit boards. They also lack standardization and have poor noise immunity. TTL bit-slice circuit elements, like the Advanced Micro Devices (AMD) 2900, for example, offer noise immunity but lack the speed needed to take advantage of the high speed memories. An alternate approach to achieving high-speed computation would be to use parallel ALU's. The Plessy MIPROC-16 computer is an example. However, to achieve reasonable speed (a 250 nanosecond microcycle time) at low cost, the MIPROC-16 incorporates a single accumulator, a 16-bit instruction word length, and an awkward addressing mode. This limits the computational power of the machine and requires many cycles for complex operations.

Thus, to provide the required speed and programming flexibility for the real-time simulator, it was necessary to custom design a processor. Making the design compatible with TTL and ECL bit-slice architectures would leave open the possibility for incorporating new bit-slice technology as it becomes available to improve performance and/or lower costs. The basic processor design is a 16-bit computer with a 32-bit instruction word length. The design incorporates saturated logic of the Schottky and low-power Schottky TTL family. Current packaging technology restricts the flexibility of available VLSI and LSI devices. Hence, our custom-designed computer features mostly medium-scale integrated (MSI) circuits in the ALU. The micro-programmable architecture employs instruction prefetch and permits pipelining of processor control signals to increase speed and provides the ability to modify the instruction set when necessary.

ARCHITECTURE OF THE CUSTOM-DESIGNED PROCESSING ELEMENT

The architecture of the custom-designed PE is presented in figure 3. It consists of an arithmetic logic unit (ALU), high-speed data busses, a high-speed TRW MPY16HJ multiplier, a status register with an associated status logic generator, an exponent generator/zero detector, a microprogram controller for sequencing execution of microcoded instructions and a 32K-by-16-bit memory. These components and associated design considerations will be discussed further in the following sections of this report. Also included in the PE architecture are a program counter, which points to the next instruction to be executed, and a memory address register, which keeps track of memory access. Associated with each of these is an adder which can be used to increment or decrement the program counter or memory address register. Access to the PE is provided by an input/output port and an external memory port. Through these paths the programmer not only may monitor parameter values being used by the PE, but he is also provided with a means of modifying those which he feels need adjusting.

The Status Register

At any instant during execution, a special register gives the current condition and mode of operation of the computer and the calculations taking place. It also provides the programmer with control over the execution of the program by allowing him to select and/or change the mode of operation. In the custom-designed computer this special register, called the status register, is 16 bits long. Its layout is shown in figure 4. Notice that by setting bits 12 through 15, the programmer can enable various levels of interrupts and/or overflow limiting. When overflow limiting is enabled, any calculation that would over-flow is restricted to its full scale value. Bits 7 through 11 are a series of flags that allow different actions to be taken depending on whether they are set or reset. Bits 5 and 6 are the overflow latch and overflow latch enable, respectively. When the overflow latch is enabled (bit 6 set), an overflow will cause the overflow latch (bit 5) to be set and remain set until it is specifically reset by an action taken by the operator or the program itself. Bit 4 acts as a carryout bit; that is, it is a storage location for a bit which may be lost in a calculation due to limited register size. For example, in a 16-bit machine, adding the hexidecimal words FFFF and FFFE (-1 plus -2 decimal) results in FFFD (-3 decimal) with the carry bit (bit 4) set because a bit is "carried out" of the calculation. The final four bits (bits 0 through 3) are designated as the condition code. Generally, for arithmetic operations bit 3 indicates whether an overflow has occurred, and bits 0 through 2 indicate whether the result is positive, zero, or negative, respectively. Generally, for logical operations comparing A with B, the condition code bits 0 through 2 indicate whether A is greater than B, equal to B, or less than B. A summary of status register characteristics is presented in table I.

Status Logic Capabilities

Comparisons. - A unique feature of the PE is that circuitry has been included in the design which allows valid, logical comparisons of arbitrarily large, multiple-precision signed-words, as well as single-precision signed-words. That logical comparisons of signed-words cannot be made by just subtracting one word from another in the ALU and checking the sign of the result is well-known. To do so can result in an error. To prevent these kinds of errors, an external zero detector circuit has been included as part of the PE hardware. The inputs to the zero detector include: (1) the words to be compared; (2) the output bits, U_{XX}, of the ALU; and (3) bit 1 of the status word, SW1, which is set when, at the time the status word is updated, all bits of the current ALU output are zero (indicated as, $U_{XX} = 0$).

The logical comparison is based on the relationship that if B is not greater than A $(\overline{B > A})$ and B is not equal to A $(\overline{ZERO})$, then B must be less than A. $\overline{ZERO}$ denotes the output from the external zero detector circuit. When performing single-precision comparisons and when operating on the least signficant 16 bits of multiple-precision words, the $\overline{ZERO}$ signal will be true if any of the U_{XX} bits output from the ALU are set. When operating on the more significant 16-bit portions of multiple-precision words being compared, $\overline{ZERO}$ represents the current status of a running test on the multiple-precision comparison. Here the $\overline{ZERO}$ signal is based not only on the current 16 bits being compared (indicated as U_{XX}), but also on the previously compared 16 bits (indicated as SW1). The $\overline{ZERO}$ signal then satisfies the relationship

$$\overline{ZERO} = \overline{SW1 \cdot (U_{XX} = 0)} \tag{1}$$

For both single and multiple-precision comparisons, the B > A signal is determined by the sign bit and the carryout bit from the ALU subtraction. Thus, the logical comparison approach used in the PE is valid for both single and multiple-precision comparisons. The multiple-precision comparisons reduce to a series of multiple-precision subtractions in the ALU together with the use of a running condition code.

Because the custom-designed PE is a research-oriented computer, it has been designed to be extremely flexible when it comes to logical operations. For example, it has been designed to permit the multiple-precision comparisons to be performed from the most significant words to the least significant words. In some applications this can result in increased speed. The general procedure is similar to that just discussed. In this case, however, the procedure can stop after comparison of the most significant words is made, provided an inequality is detected. As before, the status register is updated after each subtraction in the ALU. SW0 is used in this case to indicate if a previous word comparison determined that $B > A$. If SW1 is set, all less significant words compared thus far have been equal. And since the less significant words of multiple-precision integers are unsigned, inequality may be detected by simply observing the carryout bit of the ALU upon subtraction. If the carryout bit is reset, B is greater than A.

Multiplications. - Since the custom designed PE uses the TRW MPY16HJ multiplier which has no built in overflow indicator, logic to sense multiplication overflow, MPYOVFL, had to be included in the custom design. The only possible way to overflow in an integer (scaled fraction) multiplication on the custom-designed PE is to try to multiply minus full scale by minus full scale. In scaled fraction notation, this amounts to trying to perform the calculation $-1*-1=+1$. The result, +1, overflows the scaled fraction format, and hence, causes a multiplication overflow condition. To determine multiplication overflow all that needs to be done is to determine when minus full scale is fed to each input of the TRW multiplier. Since this multiplier is separate from the ALU, the ALU can be used to add both multiplier inputs while the multiplication is taking place. The multiplication overflow indication, then, can come from the relationship

$$\mathrm{MPYOVFL} = \mathrm{OVFL} \cdot (U_{XX} = 0) \qquad (2)$$

since, when minus full scale is added to minus full scale, both an overflow (OVFL) occurs in the ALU and the U_{XX} output of the ALU is zero. Notice that this is the only set of inputs to the ALU that will result in these conditions.

The Instruction Set

Normally, instructions reside in a computer as an integral part of its hardware. Thus, the programmer must be satisfied with the instruction set built into the computer he is using. He may be able to construct some other instructions by building PROCEDURES or MACROS using the instructions provided to him by the manufacturer. This may necessitate the use of awkward constructs that require extra execution time and more core storage. In a time-critical simulation application, this may not be acceptable. The custom-designed PE includes a microcoded instruction set which offers several advantages over a system with conventional hardwired instructions. As mentioned previously, the PE was designed for use as a research tool to develop and evaluate parallel processor hardware and software. Having the instruction set reside in software complements the versatility of the design. Instructions can be conveniently added or deleted from the instruction set as the need arises.

The instruction set residing in microcode in the custom-designed PE gives the jet engine simulation programmer extraordinary computing power with which to do his work. The highlights of this instruction set are summarized in table II. The set includes 285 instructions in all. There are 22 different arithmetic instructions. These include the basic operations of addition, subtraction, multiplication, and division operating on integer (including scaled fraction) or floating point inputs, both single precision and double precision. The enabling and controlling of the interrupt structure and data transfer control of the machine is governed by 11 different control instructions. These instructions also control input/output to the PE.

The following instructions are unique to this machine insofar as they are extremely fast and provide the programmer with powerful one instruction computing capability.

Data type conversions are provided by 22 data conversion instructions. These allow any of the data types: integer (scaled fraction), floating-point, both single and double precision, to be converted to any other data type automatically. These instructions allow automatic scaling and descaling between integer and floating-point. Implementation of integration schemes used in the simulations can be facilitated by 14 instructions which perform multiple-precision, cumulative addition and subtraction.

Block move instructions allow the moving of the contents of blocks of memory around within memory using only a single instruction. Single instructions also allow fast implementation of complex multiconditional jumps. The jump will occur on status word condition true. The status word condition is field selectable on four fields with 16 different combinations on each field.

Instructions operating on integer input data are very fast. They require at most two machine microcycles to execute, with most requiring only one. The multiply instruction using the TRW MPY16HJ multiplier requires 3 microcycles to execute and the divide instruction executes in only 19 microcycles in this custom-designed machine. Likewise, select maximum/minimum requires only one microcycle for 16-bit integers.

For comparison purposes, the time required to execute typical instructions with the custom-designed PE and various commercially available machines is presented in table III. The table shows the superior speed advantage of the PE.

The Memory and Addressing Modes

As mentioned earlier, the PE has a 32K-by-16-bit memory. Included in the PE architecture is an external memory port interface which allows access to the PE memory via external data busses (see fig. 3).

Memory addressing is aided by the powerful addressing modes which allow flexibility in specifying memory locations. Memory may be specified as a base location, as a relative displacement, as a relative location specified in an index register, or as a combination of all three. In addition, memory locations may be specified as an absolute address offset by a relative location specified in an index register.

In the base address mode of referencing memory, the memory address is obtained by adding together the contents of two registers - one known as a base register, the other known as an index register - plus a 12-bit displacement specified as a constant in the operand field of the instruction. As shown in figure 3, the contents of the base and index registers for memory calculations are transported on the RA and RB register busses, respectively. The displacement comes directly from the instruction register. The reason for two registers is to allow the addressing of a block of data. The base register holds a reference location which acts as an origin for determining the addresses of the other memory locations. The contents of the index register acts as an index, just as its name implies, allowing the programmer to address a block of memory. He may also address a pattern of memory locations by changing the contents of the index register in some prearranged fashion.

In the absolute address mode of referencing memory, the memory address is obtained by adding together a 16-bit constant and the contents of an index register. For the memory address calculation these data are transported on the RA and RB register busses, respectively. The result of this summation is an absolute memory reference relative to the first location used by the simulation. Either of these addressing modes may be used in a simulation at any time. The programmer is free to intermix them, using the one he feels is most suited to his application in that particular portion of the simulation.

Microprogram Control Logic

The microprogram control logic (MPCL) which coordinates execution of the PE microcoded instructions, can be considered as a computer within the PE. Each PE instruction has associated with it a sequence of microcode commands which carry out the desired PE operation. When the microprogram requests an operation to be performed, sufficient cycles are granted to allow that operation to be carried out. The microprogram memory is 1K-by-72-bits. Each microprogram instruction is 72 bits long. The MPCL is displayed in figure 5. The op-code for the next PE instruction is directed from the prefetch latch to the microprogram control shifter. The microprogram control shifter sets the microprogram counter to the correct microprogram address in order to begin executing the sequence of code which will perform the operation. Inputs are sent to the ALU via high-speed busses where the arithmetic operations then take place. Included in the MPCL is a prioritizer which controls the operations. It will halt other operations if it determines that: (1) the system has been master cleared, (2) transfer of data between PE's is in progress, (3) a halt has been requested, (4) a parity error has been detected, (5) the op-code for the next instruction is not available from the instruction prefetch, (6) an external interrupt has occurred, or (7) a system pause has been requested by the operator.

The select logic for the microprogram next address control is shown in figure 6. Depending on whether a special microprogram bit is set or reset in the microprogram instruction word, one of two paths is selected as shown in the figure - either tier A or tier B. Furthermore, depending on whether condition code bits cc1 and cc2 are set or not, various options can arise in selecting the address of the next microprogram instruction to be executed. For example, suppose that tier B is being followed. If cc1 is reset, then the condition for a conditional jump is not satisfied. The jump is not taken and the next microprogram instruction to be executed is the next microprogram instruction in the program sequence. However, if cc1 is set and cc2 is also set, then the condition for a jump is also satisfied. A jump is taken to the address indicated (pipeline address), and the microprogram instruction contained therein is the next microprogram instruction to be executed. If cc2 is reset, the microprogram incrementer (shown in fig. 5) advances the microprogram counter by one and the next microprogram instruction in the sequence is executed as the next instruction.

If the tier A path is followed and ccl is reset, interrupts may be serviced. If an interrupt has occurred, a jump is made to the interrupt map to determine the cause and the action to be taken. Examples include: HALT, to stop execution of the program; MASTER CLEAR, to clear registers and reinitialize the PE; a parity error having occurred; transfer of data among PE's about to take place.

The Arithmetic Logic Unit (ALU)

The ALU logical arrangement is shown in figure 7. The ALU is responsible for performing the actual arithmetic and/or logical operations requested in the instruction operation code. Inputs to the ALU come via two high-speed data busses (A-bus and B-bus in fig. 3). Output from the ALU goes to a multiplexer/shifter where the final output is shaped as to whether it is to have its bits reversed, passed as is, or shifted left or right. These values are then stored in registers and/or memory via high-speed data busses for use in future calculation, as appropriate. An overflow detector checks the output to determine whether an overflow has occurred. If overflow limiting is in effect, a result which overflows is limited to its corresponding full scale value with appropriate sign. The output also goes to the status logic generator so that the status register may be updated to reflect the calculation just completed in the ALU. Information necessary for the ALU to carry out its duties is supplied to the ALU control via high-speed data busses from the program instruction register, the microprogram control, the status register, and the PE memory.

High-Speed Data Busses

Information is moved around within the custom-designed PE structure via high-speed data busses. The layout of the bus system is shown in figure 3. The A and B-busses are used to transmit input information to the ALU. The B-bus also supplies the TRW high-speed multiplier with input information. These busses receive register information from the register busses (RA and RB busses) through a system of latches also shown in figure 3. The RA and RB busses also transmit register information for memory storage and for memory address calculation. Address information is transmitted via the memory address bus to the high-speed multiplier, to the instruction address register through the prefetch latch, and to memory.

A special pair of busses are used to transfer data between PE's in the digital parallel processor. The data being transferred is transmitted over the transfer data bus, and the address from which the data came is transmitted over the transfer address bus. The transfer control is used to signal the system when a computation cycle is completed, so that a data transfer cycle may begin.

Data Transfer Between Processing Elements

When a data transfer cycle begins, the transfer control logic increments the address register every 100 nanoseconds. To meet this high rate of data transfer between PE's, pipelining, special 45 nanosecond memory, and Schottky TTL circuitry are used.

The memory configuration during a transfer cycle is shown in figure 8. The address of the data to be transferred is transmitted to the source memory from the transfer controller via the transfer address bus. Once this command is received, the data is latched at the source memory output. Then the PE places its latched data onto the 16-bit high-speed data transfer bus. The destination PE's latch in the data and store it. The storage location of the data received is determined by the local destination address register. Before a transfer cycle occurs, each PE initializes this local register and automatically increments the memory address after receiving each 16-bit data word.

Because pipelining is used during the data transfer cycle, a data transfer to as many as nine different destination processors can take place every 100 nanosecond clock cycle. After the first cycle, which is needed to initialize the pipeline, the source PE will gate the source data onto the data transfer bus. Those PE's designated as destinations then latch the incoming data. After incrementing their local addresses, they clock in a new command word and thus prepare for the next cycle. When the transfer cycle is completed, the control logic reinitializes the address register.

Diagnostic Test Software

Several diagnostic programs have been written to check that the custom-designed PE hardware is operating properly. A complete memory test was carried out at the bit level using a generalized memory test algorithm (ref. 4). Using a similar algorithm, a complete register test at the bit level was also carried out. These diagnostics checked the hardware and the microcode software to make sure that for each register or memory location every bit could be set and cleared; and that while a bit pattern was being stored in one word, bits were not being erroneously cleared and/or set elsewhere. The system, undergoing testing, is shown in figure 9.

Once it was determined that the PE registers and memory were working properly, the checkout of the microcode software for each instruction was initiated. The philosophy followed for checking the instructions was to execute each instruction with every possible combination of the sign bit and the two most significant data bits of the instruction input parameters. The test was run once with the remaining data bits set, and then again with the remaining data bits cleared. It was felt that if a wiring error existed in the hardware or if an error existed in the microcoded instruction, it would manifest itself using this set of input parameters for the diagnostic test. The output from each instruction for each set of input data was then checked against a table of hand calculated values of predicted PE outputs. Anytime a discrepancy between the PE computed value and the table value occurred, the error was flagged. If no discrepancy occurred, using the input values described above, the instruction was considered to have passed the error test. That is, it was assumed that the instruction was microcoded correctly and that it was executing correctly.

Fifteen diagnostic programs were developed to cover the checking of the various classes of instructions in the custom PE's library. Each instruction was checked by one of the diagnostic programs. In addition to the memory and register diagnostic programs already mentioned, separate diagnostic programs covering the checking of jump instructions, shift instructions, status word modification instructions, and Boolean logic instructions were developed. The remaining diagnostic programs were used to check the arithmetic instructions.

STATUS OF THE CUSTOM-DESIGNED PROCESSING ELEMENT

At the current time, prototype hardware for the custom-designed processing element is built, and most of the instructions operating on either integers or scaled fractions are operational. The Boolean logic, shift, and jump instructions also execute satisfactorily. The arithmetic operations of addition and subtraction are operational. Problems have been experienced with the divide instructions and multiply instructions. The divide instruction executes correctly as long as both inputs are from registers. If one of the inputs is from memory, however, the result of the division is incorrect. The source of the error has not yet been found. But because the divide instruction does execute correctly in the all-register-input case, the basic divide algorithm does not seem to be the source of these errors. More than likely the problem is being caused by a hardware wiring error or incorrect microcoding.

Still to be checked out are floating-point instructions. Having floating-point instructions available would be very convenient because the programmer wouldn't have to be concerned about scaling the parameters in his engine simulation. However, floating-point capability is not considered critical in the PE. Although it takes more time for the programmer to set up a fixed-point simulation because of the required scaling, simulations using integer or scaled fraction arithmetic generally will execute faster. And as mentioned earlier, achieving high execution speed was a major consideration in designing a PE for real-time simulation of jet engines and their controls.

CONCLUDING REMARKS

This report describes a prototype of a custom-designed computer to be used as a processing element in a multi-processor-based jet engine simulator. The purpose of the custom-design was to give the computer the speed and versatility required so that it could be used as a research tool for the development and testing of airbreathing engine digital electronic controls. Speed was of utmost importance in designing this custom computer. Using state-of-the-art components and innovative circuitry, a computer computation cycle time of 133 nanoseconds was achieved.

Problem areas encountered in designing the processing element, along with logic circuitry used to eliminate those problems, are pointed out. These may give the reader some insight into the kinds of difficulties that might be expected when undertaking the development of a custom-designed computer to meet special application needs. The authors hope that this report proves helpful and facilitates that kind of development. In the future, undertakings of this kind will be prompted more and more by the rapid advances being made in the area of microelectronic technology. Powerful integrated circuitry is becoming available on more dense and less costly chips. These powerful, inexpensive chips will act like building blocks which the designer will be able to link together in many different configurations, giving him the capability of custom tailoring a computer to his exact needs.

REFERENCES

1. Mihaloew, J. R.: A Nonlinear Propulsion System Simulation Technique for Piloted Simulators. NASA TM-82600, 1981.

2. Szuch, J. R.: Advancements in Real-Time Engine Simulation Technology. NASA TM-82825, 1982.

3. Blech, R. A.; and Arpasi, Dale J.: An Approach to Real-Time Simulation Using Parallel Processing. NASA-TM 81731, 1981.

4. Milner, E. J.: A Generalized Memory Test Algorithm. NASA TM-82874, 1982.

TABLE I. - STATUS REGISTER SUMMARY

- 40 Instructions for direct manipulation of the status register
- Conditional jumps and links
  (1) Field selectable on 4 fields
  (2) 16 Combinations within each field
- Overflow flag <u>and</u> overflow latch with enable
- Auto overflow limiting with enable
- 5 Program flags - 4 with external set
- 4 Condition code bits plus a carry bit
- 3 internally vectored interrupts
- Running conditionals for multiple-precision operations

TABLE II. - HIGHLIGHTS OF MACHINE INSTRUCTION SET

- 22 Basic arithmetic instructions
- 11 control instructions
- 22 data conversion instructions
- 14 integration instructions
- 5 Addressing modes
- Block move instructions
- Complex functional jumps
- Fast 1-2 cycle integer instructions
- 3 1/3 Cycle multiply
- 19 Cycle divide
- 1 Cycle 16-bit integer select maximum/minimum

TABLE III. - COMPARISONS OF MICROPROCESSORS
FOR REAL-TIME SIMULATOR COMPUTATIONS

Basic Function		Estimated time/function (μsec)				
		7.5 MHz Lewis Custom PE	8 MHz Motorola 68000	6 MHz ZILOG Z8002	10 MHz INTEL 8086	68000/Custom PE ratio
Add/sub	16-bit variables	0.25	1.43	1.55	1.61	5.7
	16-bit constants	.133	1.0	1.17	1.3	7.5
	32-bit variables	.35	1.93	2.52	3.22	5.5
Compares	16-bit integer	.33	2.68	2.60	2.27	8.1
Mult./Div.	16-bit integer	.88	13.2	15.3	14.0	15.0
Integration	32-bit	.80	4.75	8.6	6.4	5.9
	48-bit	.93	11.0	14.5	8.9	11.8

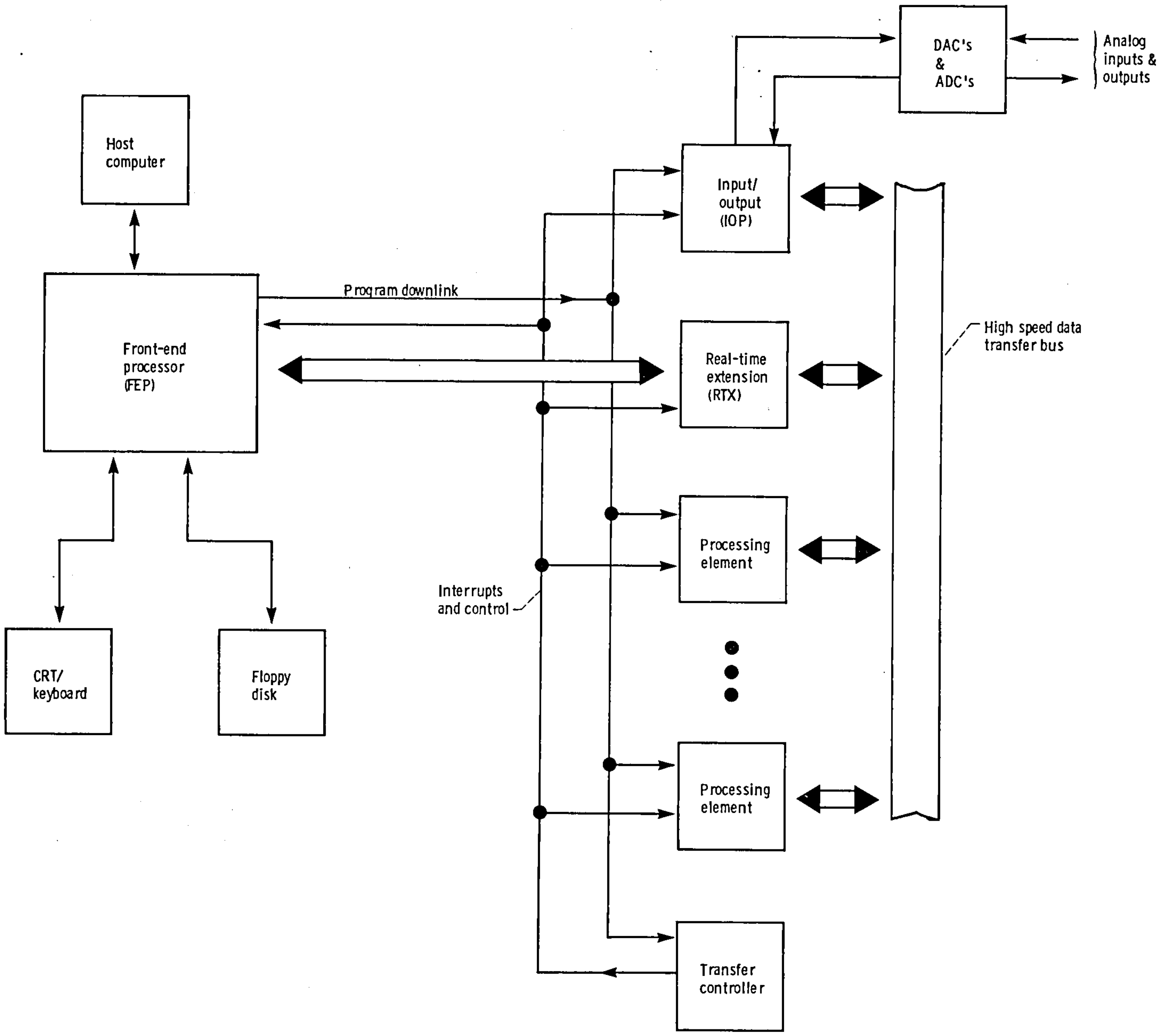
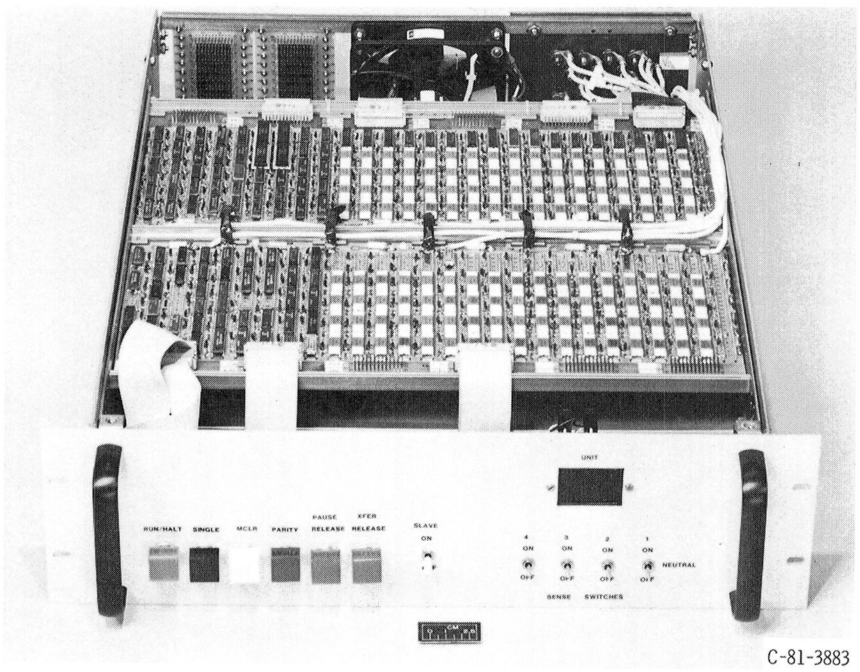


Figure 1. - Basic simulator structure.



C-81-3883

Figure 2. - LeRC custom-designed processing element.

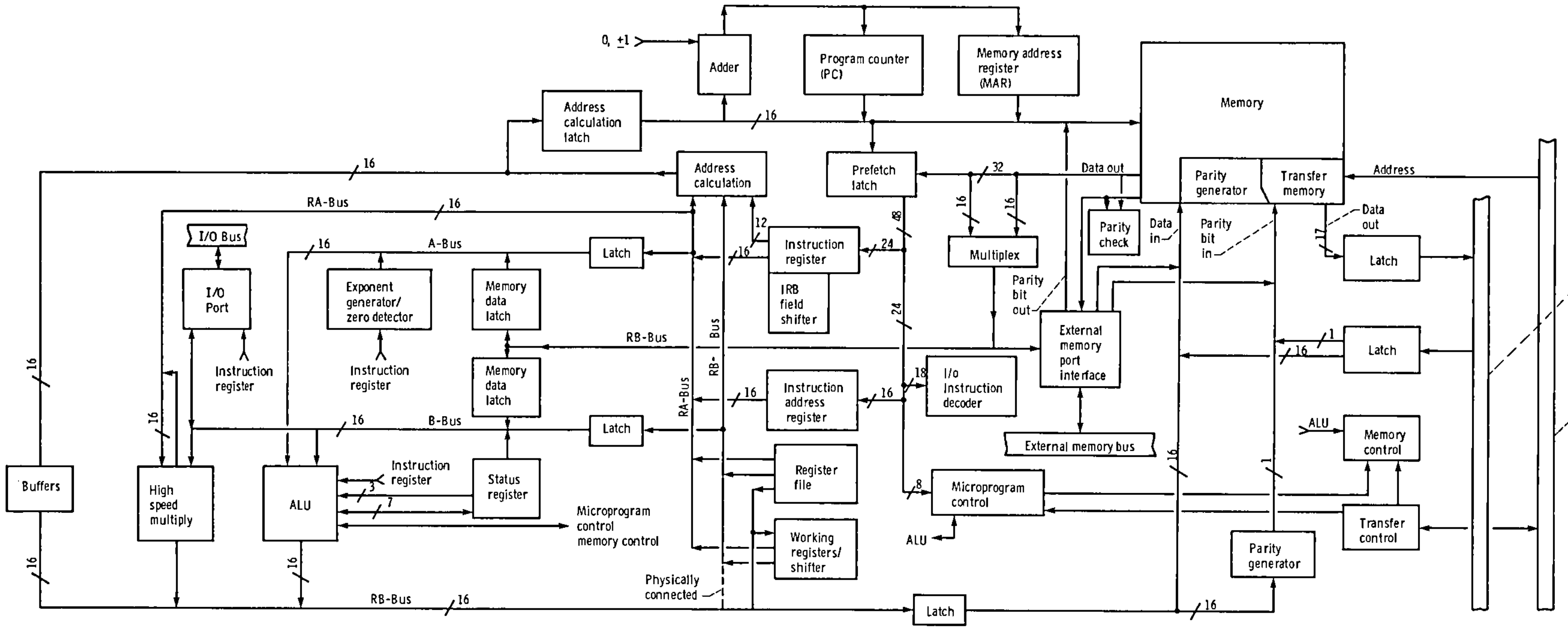


Figure 3. - Processing element architecture.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Overflow limit enable	INT 2 enable	INT 1 enable	INT 0 enable	Sensed flag 4	Sensed flag 3	Sensed flag 2	Sensed flag 1	Program flag 1	Overflow latch enable	Overflow latch	Carry bit	Condition code bit 3	Condition code bit 2	Condition code bit 1	Condition code bit 0

Condition code

Figure 4. - Status register format.

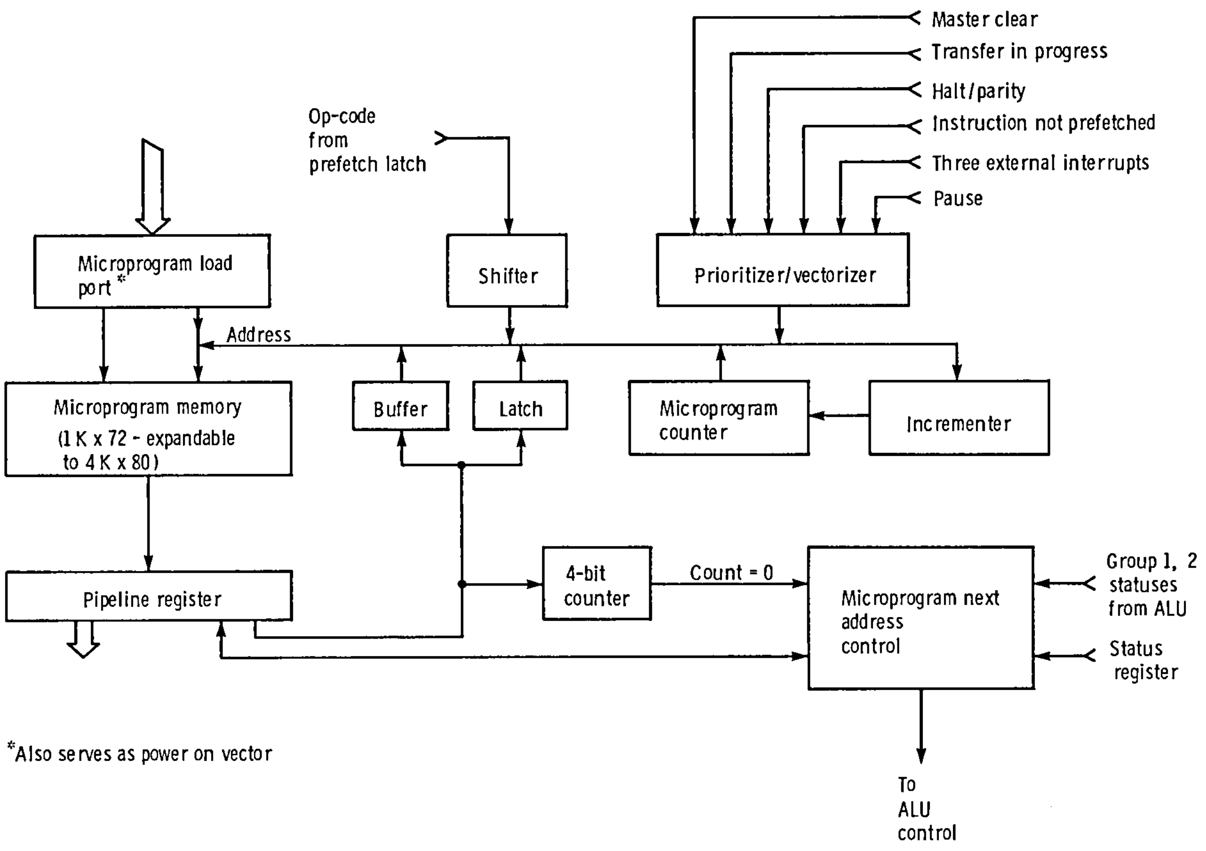

Master clear
Transfer in progress
Halt/parity
Instruction not prefetched
Three external interrupts
Pause
Op-code from prefetch latch
Microprogram load port*
Shifter
Prioritizer/vectorizer
Address
Microprogram memory (1 K x 72 - expandable to 4 K x 80)
Buffer
Latch
Microprogram counter
Incrementer
4-bit counter
Count = 0
Microprogram next address control
Group 1, 2 statuses from ALU
Status register
Pipeline register
*Also serves as power on vector
To ALU control

Figure 5. - Microprogram control logic.

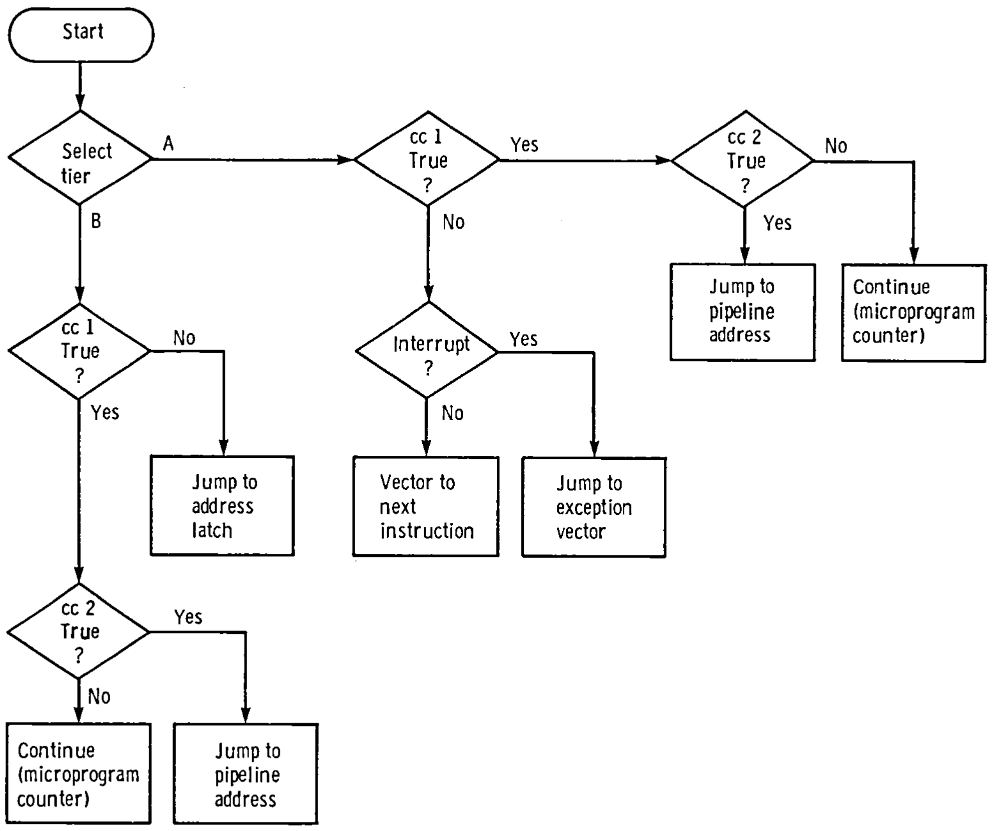


Figure 6. - Microprogram next address select logic.

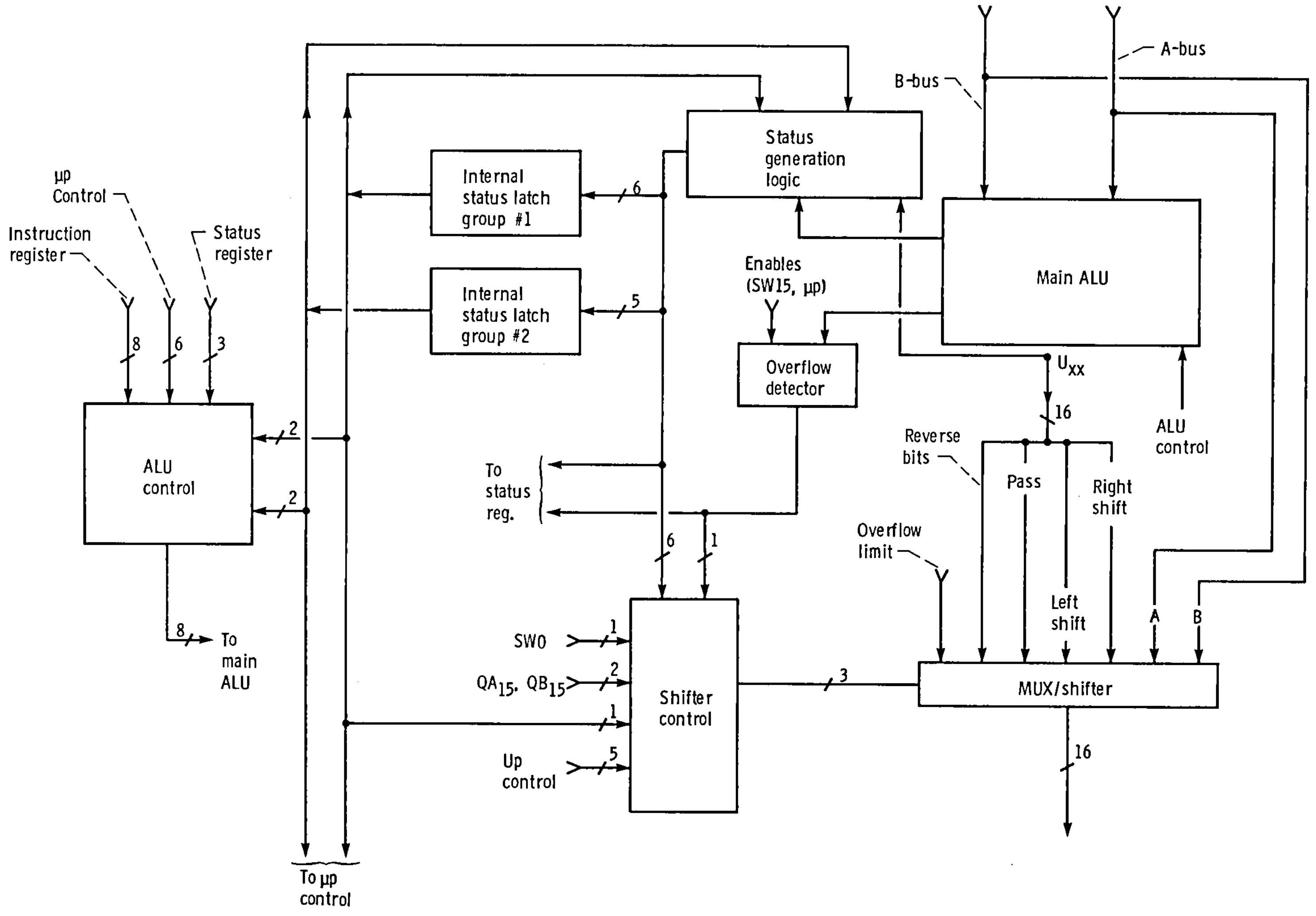


Figure 7. - ALU logical arrangement.

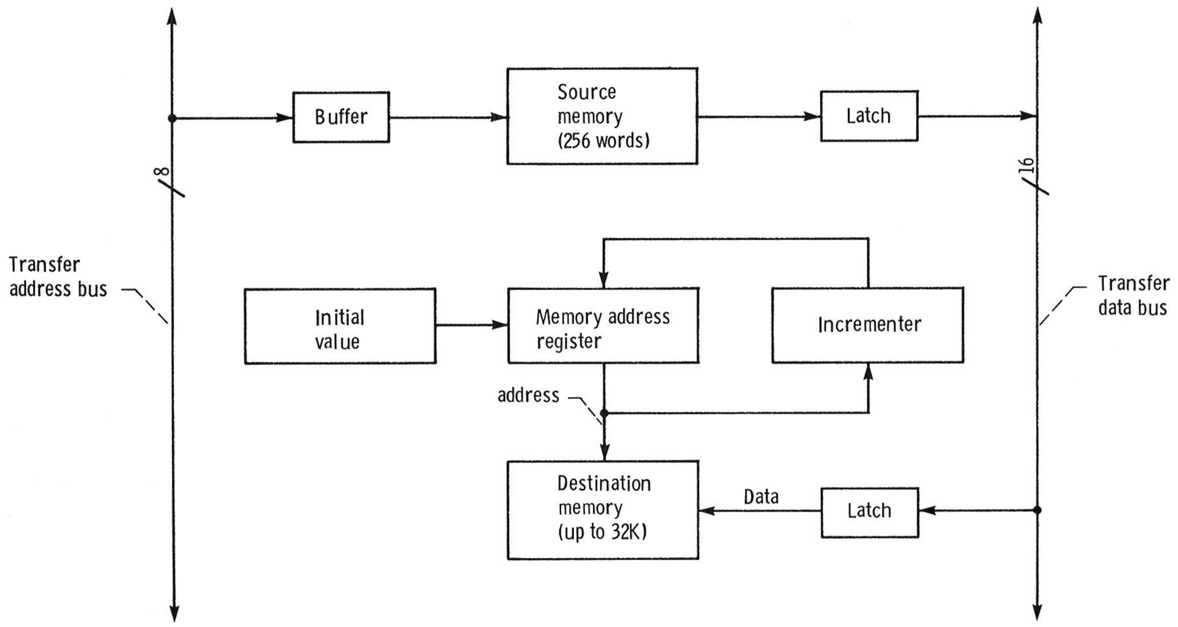


Figure 8. - Memory configuration during transfer cycle.

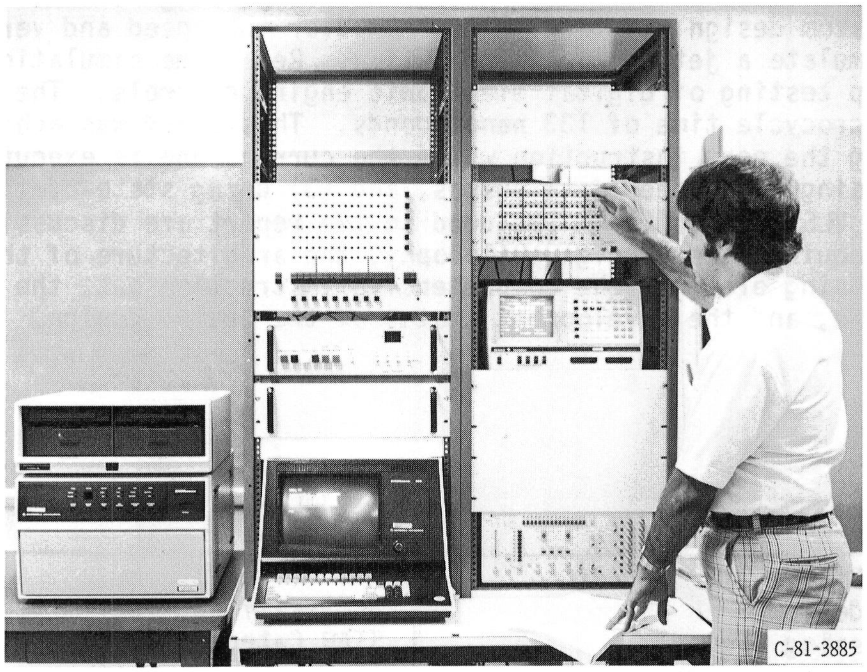


Figure 9. - Custom-designed processing element undergoing testing.

1. Report No. NASA TM-83373	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle DESIGN OF A HIGH-SPEED DIGITAL PROCESSING ELEMENT FOR PARALLEL SIMULATION		5. Report Date June 1983
		6. Performing Organization Code 505-40-5B
7. Author(s) Edward J. Milner and David S. Cwynar		8. Performing Organization Report No. E-1641
		10. Work Unit No.
9. Performing Organization Name and Address National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D. C. 20546		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

Described in this report is a prototype of a custom-designed computer to be used as a processing element in a multi-processor-based jet engine simulator. The purpose of the custom design was to give the computer the speed and versatility required to simulate a jet engine in real-time. Real-time simulations are needed for closed-loop testing of digital electronic engine controls. The prototype computer has a microcycle time of 133 nanoseconds. This speed was achieved by: (1) prefetching the next instruction while the current one is executing, (2) transporting data using high-speed data busses, and (3) using state-of-the-art components such as a VLSI multiplier. Included in the report are discussions of processing element requirements, design philosophy, the architecture of the custom-designed processing element, the comprehensive instruction set, the diagnostic support software, and the development status of the custom design.

17. Key Words (Suggested by Author(s)) Microcomputer design Parallel processing element High-speed computer design Digital simulator processor		18. Distribution Statement Unclassified - unlimited STAR Category 33	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages	22. Price*

*For sale by the National Technical Information Service, Springfield, Virginia 22161

National Aeronautics and
Space Administration

Washington, D.C.
20546

Official Business
Penalty for Private Use, $300

SPECIAL FOURTH CLASS MAIL
BOOK

U.S. MAIL

Postage and Fees Paid
National Aeronautics and
Space Administration
NASA-451

NASA

POSTMASTER: If Undeliverable (Section 158
Postal Manual) Do Not Return